Entry of this amendment is respectfully requested.

Respectfully submitted,

Dated: 1-22-02

Ralph F. Hoppin

Ralph F. Hoppin, Reg. No. 38,494
Attorney for Applicants
BROWN RAYSMAN MILLSTEIN FELDER & STEINER LLP
900 Third Avenue
New York, New York 10022
(212) 895-2000

6

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of	:	VUKOVLJAK et al.			
Application No.	:	09/928,245			
Filed	:	August 10, 2001	Group Art Unit	:	2183
Title	:	SYSTEM AND METHOD FOR TESTING MULTIPLE DIAL-UP POINTS IN A COMMUNICATIONS NETWORK			

Assistant Commissioner for Patents
Washington, D.C. 20231

Marked Up Version Of Replacement Paragraphs Showing Changes Relative To Previous Version

This paper provides a marked up version of the replacement paragraphs showing the changes relative to the previous version. The changes are shown in red ink below in photocopied pages of the specification as filed. A clean copy of the replacement paragraphs is provided in the accompanying Preliminary Amendment.

test an Internet/intranet site 170 or other communication network site or host. Advantageously, monitoring costs are reduced since a single monitoring station may test one or more sites. Multiple monitoring stations can also be deployed in concert. The site 170 may communicate via another example network 190 having other sites. In addition to testing the dial-up points, the monitoring station 100 may test services that are provided by, or via, the site 170. The monitoring station 100 may be located locally or remotely relative to the dial-up point.

The monitoring station 100 may report the test data it obtains to an object server 191. Such reports enable all SM servers to send out-of-bound and other reporting data in a user-specified format, e.g., to a Web or email location.

Various monitors analyze the test data and provide results to the operator, e.g., via an event screen or other graphical user interface (see also FIG. 2). A monitor may be provided for each of the services in the list further below. Further analysis or functionality, such as real-time fault management, and correlation with underlying network architecture, may also be provided.

The monitoring station 100 includes a Service Monitor (SM) 110, which may have a central processing unit (CPU), memory, user interface, such as a command line interface or GUI, and a properties file. The station 100 may be a workstation or other known computer device that operates using an operating system or platform, e.g., such as Solaris™/Linux® or Microsoft Windows™ (NT/2000). Software instructions may be stored locally to the station 100 for executing by the CPU in a known manner. The SM 110 may receive profile information, discussed further below, from a profiles store 130, which in turn communicates with a SM server 150. The SM 110 stores test data results in a datalog 140, which also communicates with the server 150. In one possible embodiment, the SM Server 150 may be a 100% pure Java server to support the SMs by performing datalog management and profile management. In addition, the server 150 may distribute profiles to multiple monitoring locations, and the reporting data feeds obtained from the monitored locations to individual customer locations. The profiles 130 include a database of customer monitoring requirements, e.g., the parameters associated with the dial-up point or site that are of interest to the customer. The datalogs 140 include raw performance and service availability measurements recorded by the monitors.

The SM 110 may establish connections 160 via modems 120 of the monitoring station or local host 100 and modems or dial-up points 180 associated with the Internet/intranet site or remote host 170. The connections may be established using a variety of techniques, such as those enabled using conventional analog modems, DSL modems, and ISDN terminal adapters, also sometimes referred to as ISDN modems. Thus, the term "modem" may be used to denote a terminal adapter or the like. Conventional analog modems may use, e.g., the ITU V.90 standard or CCITT V.34 standard. The term DSL is meant to encompass DSL variations, sometimes referred to as xDSL. The SM 110 communicates with each modem via respective communication ports.

In an example implementation, the monitoring station 100 may use a single computer, with a bank of sixty-four modems that dial one or more network sites.

In addition[,] to testing the availability, data rate and the like of the dial up points, the SM 110 may run a transaction monitoring process which acts like an end user of each service at the site 170 to measure the service's current status. The transaction monitor process can combine and manage a sequence of other monitors, e.g., to simulate the actions of a real user. Events are fed back into the locally managed datalogs 140, and optionally in real-time to an end user location.

The SM 100 may include a suite of proactive software monitors, each including specific executable rules, and property files. These monitors can constantly emulate a user of several different services, regularly ensuring that each service is available for access. Moreover, each monitor can be set to test the services at different time intervals. Response times, availability status and other information can be reported to the operators to allow them to maintain constantly updated views of their networks. The information obtained through monitoring the services may use the various monitors discussed further below, e.g., DHCP, HTTP, and so forth.

The invention may be implemented in conjunction with the Netcool® products available from Micromuse Inc., San Francisco, California. However, this is only one possibility, as the invention is also suitable for use with other network monitoring and testing products.

The invention delivers many tangible benefits, e.g., to any large organization hosting a Web site or conducting electronic commerce. Specifically, the invention can efficiently

RADIUS Monitor. Monitors the RADIUS. Performs a complete dial-in test against the service provider's RADIUS, checking the response time for user authentication for logon to a network site platform.

NNTP Monitor. Monitors the Internet News services. Checks to see if the service is available, and whether "new" news has been received on the monitored news groups. Both the local news process and the status of external data feeds can be reported upon.

PING/ICMP Monitor. Checks on the reachability of given IP addresses using the ICMP protocol. Can be used to measure network latency against given service-level requirements for different types of service.

Port Monitor. Allows a user-defined service, which runs on a known TCP port, to be monitored by the dynamic configuration of protocol chat scripts, which simulate a user of that service. Allows services such as a Telnet session to be quickly modeled into a global monitoring solution.

The SMs may be implemented as a suite of modular data collectors designed to run with Micromuse's flagship Netcool/OMNIbus system, available from Micromuse Inc., San Francisco, California, or with other appropriate products. Netcool/OMNIbus is a client-server application based on the Object Server, which normalizes and synchronizes these events into a common format. This allows operators to custom design service views on-the-fly. Using Boolean filters, views are created based on which events affect the availability of user-oriented services. Probes, which are passive software modules, collect event data of hundreds of applications environments, management systems, and devices.

In Netcool's hierarchical model, a service is comprised of lower-level, more granular services. These services include key IP protocols, such as those tracked by the Service Monitors. Likewise, a service level is a linear set of hierarchical services. Events pushed into the Object Server by the Probes or Service Monitors define services in terms of binary status, e.g., "Good," "Marginal," "Unknown," "Bad," etc. These are correlated with the underlying network events such as "Link Down," "Process Down," "File System Above Threshold," "Application Failure," and so on.

In summary, the SMs can provide numerous benefits, including:

The Service Monitors can collect response time and availability data at pre-defined, default intervals. The service monitor event screen 200 may display color-coded event summaries for each pre-defined service object.

1. Realtime reports. The monitors report realtime service availability and performance information to the Object Server.

2. Near realtime reports, distributed via the network. The "datalog" option of the SMs produces two types of near-realtime service-level reports, making them available over the Web or other network. These reports include simple "traffic light" type indicators and a visual data analysis tool allowing examination of trend data, e.g., using a Java applet.

3. Historical reports, typically run each night, are distributed to the network sites specified for each profile. Reports from monitors running in diverse locations can be brought together in a single location to present a global view of the performance of that service.

FIG. 3 illustrates a flowchart of a standalone dial-up process for testing dial-up points in a communication network in accordance with the present invention.

In a standalone mode, a number of dial monitor processes are started. The processes may be threads or instances, for example, depending on the platform used. Each dial monitor process parses a profile and opens a connection to a dial-up point. The specific procedure for establishing the connection will vary depending on the platform. A Solaris™/Linux™® implementation of the invention may spawn another process, such as a PPP daemon, which actually establishes a connection and a PPP link with the dial up point as a POP of the site. Such a daemon may also handle authentication with the dialed location. Alternatively, a Microsoft Windows™® (NT/2000) implementation of the invention may use a Microsoft® Remote Access Service (RAS) Application Programming Interface (API) instead of a daemon, as this provides the developer with the interface to gather all the data the service monitor requires. Moreover, the use of this API allows for other changes to the monitor architecture detailed further below.

In the standalone mode, the Windows®-based monitor obtains configuration information based on profile entries. Once profiles are read, the Dial Monitor process spawns a thread for each "dial-up" connection. Each "dial-up" connection is defined by a "modemdevice" field in a profile entry. This also exists in a Solaris™/Linux® implementation, but a modemdevice command-line option or property entry is required to relate one to the other. The binary will

be at any time running as many connection threads as there are different comms (communications) ports (as defined by "modemdevice" field entry) defined by profile entries.

A short period after the connection has been established, typically about one second (after authentication and establishing a successful connection), the dial monitor may disconnect from the dial-up point, and send data to the object server 191 and the datalog 140 regarding the connection, such as test data regarding availability, response time and other factors. This data may be provided via the dial monitor processes. For example, the dial monitor processes may provide timing data regarding the connection.

The behavior of each dial monitor process may be based on profile entries, and also controlled by "properties", which can be entered either at command-line or read in from a properties file. Each monitor is capable of supporting multiple profiles. Note that the term "process" or the like is meant to encompass instances, such as used with Solaris™/Linux® platforms, as well as threads, such as used in Windows platforms. To enable multi-modem support, the dial monitor uses an additional property, "modemdevice", which is also mirrored in an additional field in the dial monitor profiles. This is the case for the Solaris™/Linux® based implementation. For the Windows® based implementation, the "modemdevice" property is not needed or used.

If a command line property, e.g., modemdevice, is related to an entry field in a profile, then if that property is invoked, only profiles that have corresponding entries will be run by that process of the monitor. So, any instance of the dial monitor will read profiles and only start a dial-up connection for the profile entry where the value of the "modemdevice" field matches the value of the "modemdevice" property that was used when starting this instance of the dial monitor. For example, the following code may be used to read all the profiles, finds an entry that has "modemdevice" field value "/dev/cua/b", and open that connection through the designated communications port.

./nco_m_dial –modemdevice /dev/cua/b -> monitor

On Windows®, the situation differs in that, as there is a single binary, profiles are read/parsed and a connection thread is created for each "modemdevice" profile entry. There is no need to try and synchronize multiple instances of the dial monitor with corresponding profile entries.

Each dial monitor process can establish a connection to one dial-up point at any one time. Moreover, multiple processes of the dial monitor can be run with different values for their "modemdevice" property to enable multiple dial monitor processes to establish respective different connections substantially at the same time. Each of these processes can test the profiles that have a corresponding value in their "modemdevice" field. For example, to test connectivity via three different ports, the following could be entered at the command line for a Sun Solaris ™ platform implementation:

./nco_m_dial –modemdevice /dev/cua/a –datalog &

./nco_m_dial –modemdevice /dev/cua/b –datalog &

./nco_m_dial –modemdevice /dev/cua/c –datalog &

Each of these processes may run (sequentially, if there is more than one) against a profile entry that has a corresponding value in its "modemdevice" field.

Or, depending on the platform that is used, a properties file could be used instead of the command line entries. For example, a command line interface may be used with a Unix® platform, while a properties file may be used with a Windows® platform.

It should be appreciated that various operating systems may be used, including, e.g., Sun Solaris™, Linux®, Windows NT® and Windows 2000®. The code is used to command a communications port, and as understood by those skilled in the art, can vary based on the operating system, specific equipment used, and so forth.

In the above Solaris™-specific syntax, /dev/cua/a, etc. are just port names, which will be different, e.g., on Linux®. A Windows®-based implementation of the invention, in one possible implementation, does not need this, but may use a syntax as follows:

./nco_m_dial –modemdevice <comms port 1> –datalog &

./nco_m_dial –modemdevice <comms port 2> –datalog &

./nco_m_dial –modemdevice <comms port 3> –datalog &

where <....> indicates changeable syntax.

The approach presented provides multi-modem support in the dial monitor standalone mode, and is part of the solution for the transaction mode.

The standalone process is summarized at blocks 300 through 360 in FIG 3. The steps shown need not necessarily be performed in the order given. The multiple dial monitor processes are started (block 300) sequentially, concurrently, or in any other way. For

example, for a Windows® platform based implementation, the processes may be threads that are started substantially in parallel. The dial monitor processes establish respective connections to dial up points of a network site using the designated communication port (block 320).

The dial monitor processes may be authenticated by the remote site (block 330). After the dial monitor processes obtain test data regarding their respective connections (block 340), they disconnect (block 350), and send their test data to the object sever and datalog (block 360). The test data may include data derived from the PPP daemon, when used.

FIG. 4 (comprising Figures 4-1 and 4-2) illustrates a flowchart of a transaction monitor process for testing services of a network site, in accordance with the present invention.

In the transaction mode, the dial monitor processes are used in a transaction profile, and are started or spawned as child processes of a transaction monitor process. For example, the dial monitor processes may be started when a dial-up step in the transaction monitor process is reached. The dial-up step may be the first step in the transaction monitor process as these transactions generally aim to test services over a dial up link. As mentioned, the transaction monitor process can combine and manage a sequence of other monitors, e.g., to simulate the actions of a real user.

When the connection is established, other steps in the transaction can use the connection to test services of a site. The connection may be brought down when a dial-down step in the transaction monitor process is reached. The link may be brought down earlier if the transaction monitor process has failed at any step prior to the dial-down step. The network operator may define the conditions for failure. For example, for a transaction that is designed to connect with multiple sites, an inability to connect to a site may trigger a failure.

Moreover, for testing multiple transactions/services involving the connections, each connection will have its own discrete route over which the other steps in the transaction monitor process will test services. This ensures that data gathered via a connection is representative of that connection dial-up link.

After the transaction monitor process spawns the dial monitor processes as child processes in its dial-up step, it informs them that it is running in the transaction mode. This may be achieved, e.g., by passing command-line arguments to the dial monitor processes.

The same applies to a Windows® implementation as to a Solaris™/Linux® implementation, but with Windows®, "modemdevice" property is not passed. Thus, the transaction monitor process is now modified to pass an additional command-line argument – "modemdevice"- with a value that corresponds to the "modemdevice" profile entry in the dial-up step.

At this point, testing of multiple concurrent transactions/services involving a dial-up connection is already possible. However, a problem may arise when the steps inside the transaction monitor process start testing services over the dial-up connection. When opening a connection to the remote host (e.g., network site), all other service-specific monitors in the transaction monitor processes allow the operating system to choose which local interface to connect from. This can lead to monitors that monitor the wrong dial-up link - not using the dial-up link that was opened for them, but another dial-up link that is also active at the time.

To solve this problem, the transaction monitor process can obtain an address such as the Internet Protocol (IP) address of the local host interface associated with a successful connection (dial monitor dial-up step). For example, this may be a PPP interface associated with a successful PPP link-up. The transaction monitor process also obtains the routing information associated with the connection and passes all this information to the service-specific monitors that are used in the further transaction steps. This allows the service-specific monitors to bind to the interface with the IP address that has been provided by the transaction monitor process, and create and use a route (using Operating System calls) between this IP address and the remote host being tested.

The transaction monitor has analogous functionality (as it appears to the user) as the Solaris™/Linux® version. One difference is that, when a dial process is spawned by the transaction monitor process, it will then spawn a connection thread (in Dial UP), or will signal the running dial monitor to shut down (on Dial Down). In a Solaris™/Linux® implementation, the dial monitor process spawns a PPP daemon process (on Dial UP) or signals the running PPP daemon process to shut down (on Dial Down).

The transaction monitor process is summarized in FIG. 4. Once the transaction monitor process is started (block 400), the dial monitor processes are spawned (block 410). At block 420, the dial monitor processes are informed that the transaction mode is running (block 420). This can be achieved in different ways depending on the platform.

Respectfully submitted,

Dated: 1-22-02

Ralph F. Hoppin

Ralph F. Hoppin, Reg. No. 38,494
Attorney for Applicants
BROWN RAYSMAN MILLSTEIN FELDER & STEINER LLP
900 Third Avenue
New York, New York 10022
(212) 895-2000